(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) **EP 3 642 736 B1**

(12) **EUROPEAN PATENT SPECIFICATION**

(45) Date of publication and mention of the grant of the patent:
10.01.2024 Bulletin 2024/02

(21) Application number: **18738093.6**

(22) Date of filing: **14.06.2018**

(51) International Patent Classification (IPC):
G06F 16/00 *(2019.01)* ***G06F 16/245*** *(2019.01)*
G06F 16/2453 *(2019.01)*

(52) Cooperative Patent Classification (CPC):
G06F 16/24532; G06F 16/24569

(86) International application number:
PCT/US2018/037627

(87) International publication number:
WO 2018/236672 (27.12.2018 Gazette 2018/52)

(54) PARALLEL COMPUTE OFFLOAD TO DATABASE ACCELERATOR

PARALLELBERECHNUNGSAUSLAGERUNG AUF EINEN DATENBANKBESCHLEUNIGER

DÉLESTAGE DE CALCUL PARALLÈLE À UN ACCÉLÉRATEUR DE BASE DE DONNÉES

(84) Designated Contracting States:
AL AT BE BG CH CY CZ DE DK EE ES FI FR GB GR HR HU IE IS IT LI LT LU LV MC MK MT NL NO PL PT RO RS SE SI SK SM TR

(30) Priority: **23.06.2017 US 201715632082**

(43) Date of publication of application:
29.04.2020 Bulletin 2020/18

(73) Proprietor: **Xilinx, Inc.**
San Jose, California 95124 (US)

(72) Inventors:
- **VERMA, Hare, K.**
  San Jose, CA 95124 (US)
- **SANTAN, Sonal**
  San Jose, CA 95124 (US)
- **WU, Yongjun**
  San Jose, CA 95124 (US)

(74) Representative: **Zimmermann & Partner Patentanwälte mbB**
Postfach 330 920
80069 München (DE)

(56) References cited:
US-A1- 2012 117 027
US-A1- 2012 259 843
US-A1- 2013 117 305
US-A1- 2015 046 486

- **SALAMI BEHZAD ET AL: "AxleDB: A novel programmable query processing platform on FPGA", MICROPROCESSORS AND MICROSYSTEMS, vol. 51, 28 April 2017 (2017-04-28), pages 142-164, XP085072087, ISSN: 0141-9331, DOI: 10.1016/J.MICPRO.2017.04.018**
- **MUELLER RENE ET AL: "Glacier a query-to-hardware compiler", USER INTERFACE SOFTWARE AND TECHNOLOGY, ACM, 2 PENN PLAZA, SUITE 701 NEW YORK NY 10121-0701 USA, 6 June 2010 (2010-06-06), pages 1159-1162, XP058519627, DOI: 10.1145/1807167.1807307 ISBN: 978-1-4503-4531-6**
- **DANIEL ZIENER ET AL: "FPGA-Based Dynamically Reconfigurable SQL Query Processing", ACM TRANSACTIONS ON RECONFIGURABLE TECHNOLOGY AND SYSTEMS, ACM, US, vol. 9, no. 4, 22 August 2016 (2016-08-22) , pages 1-24, XP058275630, ISSN: 1936-7406, DOI: 10.1145/2845087**
- **BECHER ANDREAS ET AL: "Energy-aware SQL query acceleration through FPGA-based dynamic partial reconfiguration", 2014 24TH INTERNATIONAL CONFERENCE ON FIELD PROGRAMMABLE LOGIC AND APPLICATIONS (FPL), TECHNICAL UNIVERSITY OF MUNICH (TUM), 2 September 2014 (2014-09-02), pages 1-8, XP032662501, DOI: 10.1109/FPL.2014.6927502**

Processed by Luminess, 75001 PARIS (FR)

(Cont. next page)

- **Tavinor Edouard: "Pipelining for FPGAs. I thought I'd write an explanation of... | by Edouard Tavinor | Medium", , 30 March 2017 (2017-03-30), XP093017999, Retrieved from the Internet: URL:https://ehrt74.medium.com/pipelining-for-fpgas-9342d3a2fdf8 [retrieved on 2023-01-26]**
- **SALAMI BEHZAD ET AL: "AxleDB: A novel programmable query processing platform on FPGA", MICROPROCESSORS AND MICROSYSTEMS, vol. 51, 28 April 2017 (2017-04-28), pages 142-164, XP085072087, ISSN: 0141-9331, DOI: 10.1016/J.MICPRO.2017.04.018**

Description

TECHNICAL FIELD

[0001] Examples of the present disclosure generally relate to a database accelerator, and in particular, to preparing and executing data blocks related to a database query in the database accelerator.

BACKGROUND

[0002] Real-time analytics are becoming increasingly compute intensive as the amount of data to be analyzed grows. This results in massive compute node scaling, which is expensive, and sometimes is infeasible. Querying and analyzing data stored in a database becomes more complicated and time consuming as the size of the database increases. Typically, a database management system (DBMS) uses one or more CPUs to perform a client query on a database. However, the amount of data that can be processed by the CPUs in parallel is limited.

[0003] Salami describes a programmable query processing platform on FPGA in SALAMI BEHZAD ET AL: "AxleDB: A novel programmable query processing platform on FPGA", MICROPROCESSORS AND M ICROSYSTEMS. vol. 51, pages 142-164, XP085072087, ISSN: 0141-9331, DOI: 10.101 6/J.MICPRO.201 7.04.018.

[0004] Mueller describes a query-to-hardware compiler in MUELLER RENE ET AL: "Glacier a query-to-hardware compiler", USER INTERFACE SOFTWARE AND TECHNOLOGY, ACM, PENN PLAZA, SUITE 701 NEW YORK NY 10121-07A1 USA, 6 June 2010 (2010-06-06), pages 159-1 162.

[0005] Ziener describes a FPGA-Based dynamically reconfigurable SQL query processing in DANIEL ZIENER ET AL: "FPGA-Based Dynamically Reconfigurable SQL Query Processing", ACM TRANSACTIONS ON RECONFIGURABLE TECHNOLOGY AND SYSTEMS, ACM, US, vol. 9, no. 4,22 August 2016 (201 6-08-22), pages 1 -24, xP058275630, ISSN: 1 936-7406. DOI: 10.1 1 4512845087.

[0006] Becher describes an energy-aware SQL query acceleration in BECHER ANDREAS ET AL: "Energy-aware SQL query acceleration through FPGA-based dynamic partial reconfiguration", 2014 24TH INTERNATIONAL CONFERENCE

[0007] ON FIELD PROGRAMMABLE LOGIC AND APPLTCATIONS (FPL), TECHNICAL UNIVERSITY OF MUNICH (TUM), 2 September 2014 (201 4-09-02), pages 1 -8, XP03266250'1, DOI : 1 0. 1 1 09/FPt.201 4.6927 502.

[0008] Edouard describes pipelining for FPGAs in Tavinor Edouard: "Pipelining for FPGAs. I thought I'd write an explanation of... by Edouard Tavinor I Medium", 30 March 2017 (2017-03-30), XP093017999, Retrieved from the Internet: URL: https://ehrt74.medium.com/pipeliningfor-fpgas-9342d3a2tdl8 [retrieved on 2023-01-26].

SUMMARY

[0009] Techniques for operating a database accelerator according to claim 1 are described. The invention is defined by the independent claims to which reference should be made.

BRIEF DESCRIPTION OF THE DRAWINGS

[0010] So that the manner in which the above recited features can be understood in detail, a more particular description, briefly summarized above, may be had by reference to example implementations, some of which are illustrated in the appended drawings. It is to be noted, however, that the appended drawings illustrate only typical example implementations and are therefore not to be considered limiting of its scope.

Fig. 1 illustrates a data center that includes a database accelerator, according to an example.

Fig. 2 is a flowchart for executing a query using a database accelerator, according to an example.

Fig. 3 illustrates a DBMS for offloading queries to a database accelerator, according to an example.

Fig. 4 illustrates converting a database table to a data block customized for parallel computation, according to an example.

Fig. 5 illustrates performing parallel tasks in a database accelerator, according to an example.

[0011] To facilitate understanding, identical reference numerals have been used, where possible, to designate identical elements that are common to the figures. It is contemplated that elements of one example may be beneficially incorporated in other examples.

DETAILED DESCRIPTION

[0012] Various features are described hereinafter with reference to the figures. It should be noted that the figures may or may not be drawn to scale and that the elements of similar structures or functions are represented by like reference numerals throughout the figures. It should be noted that the figures are only intended to facilitate the description of the features. They are not intended as an exhaustive description of the description or as a limitation on the scope of the claims. In addition, an illustrated example need not have all the aspects or advantages shown. An aspect or an advantage described in conjunction with a particular example is not necessarily limited to that example and can be practiced in any other examples even if not so illustrated, or if not so explicitly described.

[0013] Embodiments herein describe techniques for preparing and executing tasks related to a database query in a database accelerator. In one embodiment, the

database accelerator is separate from a host CPU. For example, the database accelerator may be implemented using a programmable integrated circuit (e.g., a field-programmable gate array (FPGA)) or a non-programmable integrated circuit (e.g., an application specific integrated circuit (ASIC)). In any case, the database accelerator is communicatively coupled to the host CPU which executes a database management system (DBMS).

[0014] The DBMS can offload tasks corresponding to a database query to the database accelerator. The DBMS requests data from the database relevant to the query and then converts that data into a data block that is suitable for processing by the database accelerator. The database accelerator contains individual hardware processing units (PUs) that can process data in parallel or concurrently. However, in order to process the data concurrently, the DBMS converts the database tables retrieved from the database into a data block that includes individual PU data blocks that are each intended for a respective PU in the database accelerator.

[0015] The PUs in the database accelerator execute the PU data blocks in parallel. For example, the PUs may analyze the data as dictated by the client query (e.g., a structured query language (SQL) query) and then return individual results. The database accelerator combines the individual results and returns the combined results to the DBMS. In turn, the DBMS may merge the results received from the database accelerator with other data before transmitting query results to the client. In this manner, the DBMS can take advantage of the parallelization performed by the PUs in the database accelerator to execute a database query.

[0016] Figure 1 illustrates a data center 100 (e.g., a query processing system) that includes a database accelerator 145, according to an example. The data center 100 includes databases 105A and 105B and a compute node 110. In one embodiment, the databases 105 are relational databases where the data stored in the databases 105 can be accessed by SQL queries. However, the embodiments herein are not limited to relational databases.

[0017] The compute node 110 includes a host CPU 115 and a FPGA 140 that are communicatively coupled via a PCIe connection 130. The host CPU 115 executes a DBMS 120 (e.g., a software application executing on the host CPU 115) that can offload tasks related to performing and analyzing a client query to the database accelerator 145. In one embodiment, the DBMS 120 is a relational DBMS (RDBMS). Although not shown, the DBMS 120 receives queries (e.g., client queries) to execute on the databases 105. For example, the queries may be a search, sort, word count, or query 6 which instruct the DBMS 120 to retrieve data from the databases 105, process the data, and then return a result to the client that transmitted the query. Moreover, the DBMS 120 may generate updated data which is then stored in the databases 105.

[0018] Instead of using the host CPU 115 to execute the query, the DBMS 120 offloads at least a portion of the task (or workload) corresponding to the query to the database accelerator 145 in the FPGA 140. In one embodiment, the database accelerator 145 may do a portion of the tasks associated with the query while the host CPU 115 performs the remaining tasks. The DBMS 120 can merge the results obtained from the database accelerator 145 with the results generated by the host CPU 115.

[0019] The FPGA 140 includes programmable logic in one or more programmable integrated circuits (not shown) which is configured to implement the database accelerator 145. One advantage of using an FPGA is that its programmable logic can be configured using a hardware description language into PUs 150 which can operate in parallel. That is, the FPGA 140 may be better suited for performing a large number of tasks (or threads) in parallel. Although an FPGA 140 is shown, in another embodiment the database accelerator 145 can be implemented on a non-programmable integrated circuit such as an ASIC. If an ASIC is used, the ASIC can be designed to include PUs 150 for performing tasks in parallel. As such, the embodiments described herein are not limited to a database accelerator 145 implemented in programmable integrated circuits.

[0020] To share data with the FPGA 140, the DBMS 120 contains an accelerator library 125 which include functions for moving data between the host CPU 115 and the FPGA 140. In one embodiment, the accelerator library 125 includes a conversion function (referred to herein as an enqueue compute) which converts the data received from the databases 105 into a format that can be processed in parallel by the database accelerator 145. The data received from the databases 105 is organized in database tables or database pages where data is arranged in rows and columns. Using the enqueue compute function, the DBMS 120 converts the database tables into a format which can be assigned to the various PUs 150 in the database accelerator 145. In addition to the enqueue compute function, the accelerator library 125 may include other commands for writing, reading, and merging the data transmitted to, and received from, the database accelerator 145.

[0021] The PCIe connection 130 permits the host CPU 115 and the FPGA 140 to share data, however, any high speed connection can be used. In one embodiment, the DBMS 120 can perform a direct memory access (DMA) to transmit data to and from the database accelerator 145.

[0022] In one embodiment, the compute node 110 is a computing device such as a server. In another embodiment, the compute node 110 is a blade server. Alternatively, the compute node 110 may be a separate component in a computing device. For example, the compute node 110 may be a common substrate on which the host CPU 115 and the FPGA 140 are mounted - e.g., a motherboard or a field replaceable unit (FRU). In another arrangement, the host CPU 115 may be disposed on a motherboard while the FPGA 140 may be plugged into

an expansion slot in the motherboard. Thus, the FPGA 140 can be added to the compute node 110 in order to increase the ability of the host CPU 115 to execute database queries. In another example, the host CPU 115 and the FPGA 140 can be packaged together.

[0023] In one embodiment, the client query is submitted to the data center 100 and is assigned to the DBMS 120. For example, the data center 100 may be a part of a cloud computing environment that stores client data in the databases 105. As described in more detail below, the DBMS 120 determines how much (if any) of the client query should be offloaded to the database accelerator 145. The DBMS 120 retrieves the relevant data from the databases 105, analyzes the data with the assistance of the database accelerator 145, and transmits results back to the client. Although the databases 105 and compute node 110 are shown as being co-located in the data center 100, in other embodiments, the databases 105 may be disposed at a different geographic location than the compute node 110.

[0024] Figure 2 is a flowchart of a method 200 for executing a query using a database accelerator, according to an example. For clarity, the blocks in the method 200 are described in parallel with Figure 3 which provides a more detailed illustration of the functions and components of the DBMS 120 and the database accelerator 145.

[0025] At block 205, the DBMS 120 receives a client query to read, analysis, or update data stored in a database. The client query can originate from a person (e.g., an accountant performing payroll or a human resource manager who wants to identify employees who have a common attribute) or from a software application. In one embodiment, the client query is an SQL query.

[0026] The DBMS 120 uses the client query to transmit a database query to the database 105. As shown in Figure 3, the DBMS 120 includes a database interface 310 that permits the DBMS 120 to transmit a request for data to the database 105. For example, the DBMS 120 may include an index corresponding to the database 105 which indicates where and what data is stored in the database 105. The DBMS 120 can use the index to identify what data is relevant to the client query and then request that data from the database 105. In response to the request or query, the database 105 returns database tables 305 or database pages to the DBMS 120 via the database interface The database tables 305 arrange the data in rows and columns.

[0027] At block 215, the DBMS 120 reformats the database tables 305 into data blocks 340 that each has multiple PU data blocks that correspond to the PUs 365 in the database accelerator 145. The data blocks 340 have a format that is more suitable to parallel processing than the format used by the database tables 305. The specific format of the data blocks 340 is described later in this disclosure.

[0028] As shown in Figure 3, the DBMS 120 uses the enqueue compute function 325 from the accelerator library to reformat and queue the data blocks 340. In this example, the enqueue compute function 325 uses the database tables 305 (which can be stored in a buffer 315 in the database interface 310) to generate the data blocks 340. The DBMS 120 may use different enqueue compute functions 325 for generating data blocks 340 for different queries which are represented by the different vertical queues shown in Figure 3. In one embodiment, the different vertical queries represent different threads. Executing different threads in the DBMS 120 may avoid overloading the host CPU.

[0029] In one embodiment, multiple data blocks 340 correspond to the same query. For example, to perform a query 6, the DBMS 120 may call the enqueue compute function 325 multiple times in order to generate multiple data blocks 340 that are processed by the database accelerator 145.

[0030] The data blocks 340 may include one or more database tables 305. For example, a data block 340 may contain all or a subset of the data in a single database table 305, or a data block 340 may contain data from multiple database tables 305. In another example, a single database table 305 may be divided into multiple data blocks 340 in which case the data blocks 340 are queued in the DBMS 120 as shown in Figure 3 waiting processing by the database accelerator 145. In one embodiment, the queued data blocks 340 are sent one-by-one to the database accelerator 145.

[0031] Before transmitting the data blocks 340 to the database accelerator 145, the DBMS can perform a write callback function 330 on the data blocks 340. The write callback function 330 may be part of the accelerator library and can be performed before using a DMA to transfer the data blocks 340 to the database accelerator 145. The write callback function 330 is used to write data to the data blocks 340 before the blocks 340 are processed by the database accelerator 145. For example, the DBMS 120 may have received updated data from the client as part of the query, and as such, first writes that the updated data into the data blocks 340 before the blocks are analyzed by the PUs 365 in the database accelerator 145.

[0032] At block 220, the DBMS 120 transmits the data block 340 to the database accelerator 145. The DBMS 120 includes a FPGA framework 345 that serves as an interface between the DBMS 120 and the FPGA 140. In one embodiment, the FPGA framework 345 translates commands and instructions issued by the DBMS 120 into commands that can be interpreted by the FPGA 140. The FPGA framework 345 (e.g., a programmable integrated circuit framework) can be configured using a development environment which uses application code (e.g., OpenCL, C, C++, etc.) to enable the host CPU (e.g., an x86 based platform) to communicate with the FPGA 140. For example, the FPGA framework 345 permits the FPGA to be used as an accelerator - e.g., a database accelerator 145. One non-limiting example of a development environment for generating the FPGA framework 345 is Xilinx's SDx development environment which sup-

ports OpenCL, C, and C++ kernels. The FPGA framework 345, as well as the accelerator library 125 in Figure 1, enables the DBMS 120 to offload tasks to the FPGA 140 and the database accelerator 145 similar to a host CPU offloading tasks to other accelerators such as a graphics accelerator or crypto accelerator.

[0033] At block 225, the database accelerator 145 processes the respective PU data blocks in the data block 340 in parallel using the PUs 365 according to the query. The data blocks 340 are received at a parser 360 in the database accelerator 145. The parser 360 may divide a data block 340 into separate PU data blocks which are then forwarded to a respective one of the PUs 365. As shown, the PUs 365 contain a respective memory element - e.g., BRAM 370 - for storing the PU data blocks as well as the results of processing the data blocks.

[0034] In addition to transmitting the data blocks 340, the DBMS 120 can also transmit instructions to the database accelerator 145 indicating how the PUs 365 should process the data blocks 340 according to the received client query. For example, if the query is a word count query, the DBMS 120 transmits to the database accelerator 145 the particular word specified by the client which is to be counted. The word is then sent to the PUs 365 which traverse their respective PU data blocks to identify any occurrences of that word. In another example, if the query is a sort, the DBMS 120 provides parameters for sorting the data blocks 340 which the PUs 365 then use to rearrange the data blocks 340. In this manner, the DBMS 120 can provide both the data to be analyzed as well as instructions or parameters for analyzing the data to the database accelerator 145.

[0035] The DBMS 120 includes an event manager 320 which determines when to transmit the data to the FPGA 140 using the FPGA framework 345. For example, the event manager 320 may monitor the queues containing the data blocks 340 and determine when a data block 340 (or blocks) should be transmitted to the database accelerator 145.

[0036] The PUs 365 analyze their respective blocks according to the parameters defined by the query and/or the DBMS 120. As mentioned above, the PUs 365 are separate hardware elements, and thus, can execute in parallel. For example, the PU 365A can analyze its corresponding PU data block at the same time the PU 365B analyzes its PU data block. Although only a few PUs 365 are shown in Figure 3, the FPGA 140 can be configured such that the database accelerator 145 has tens or hundreds of PUs 365 each of which can operate independently of the other PUs 365 such that the data block 340 can be processed in parallel.

[0037] The PUs 365 forward the results of analyzing or processing the PU data blocks to a combiner 375. At block 230, the combiner 375 combines the results outputted by the PUs. For example, for a sort query, each PU 365 sorts its individual PU data blocks and then the combiner 375 may combine the sorted data block into a contiguous block of data. For a word count or search query, the combiner 375 may identify the total word count or combine the search results to identify all the locations in the data block 340 where a search term was found.

[0038] When the database accelerator 145 completes processing a received data block 340, the FPGA 140 informs the event manager 320 which executes a read callback function 335 which uses the FPGA framework 345 to instruct the FPGA 140 to transmit the results obtained by the database accelerator 145. At block 235, the FPGA 140 transmits the combined results to the host CPU and the DBMS 120 in response to the read callback function 335.

[0039] In one embodiment, although not shown, the DBMS 120 may call other functions to reformat the combined results received from the database accelerator 145. For example, if the data is to be stored in the database 105 (e.g., the database accelerator 145 changed or rearranged the data), the DBMS 120 may format the data into the format used by the database tables 305.

[0040] At block 240, the DBMS 120 merges the results received from the database accelerator 145. For example, the DBMS 120 may have divided a database table 305 into multiple data blocks 340 which were sequentially processed by the databased accelerator 145. As the results from processing the data blocks 340 are received from the database accelerator 145, a merger 355 in the DBMS 120 may merge the results from sequentially processing the individual data blocks 340 into a combined result set that corresponds to the database table 305 rather than just the individual data blocks 340. Moreover, the merger 355 can also merge the results returned from the database accelerator 145 with result provided by a local query manager 350. For example, the DBMS 120 may perform some task related to a query using the local query manager 350 and other tasks for that same query using the database accelerator 145. The merger 355 can merge the results generated locally on the host CPU using the query manager 350 and the results generated by the database accelerator 145. However, in other embodiments, all the tasks associated with a query may be performed by the database accelerator 145. In that case, the merger 355 would not need to merge the results provided by the database accelerator 145 with local results.

[0041] At block 245, the DBMS 120 forwards the query results 380 to the client. The query results 380 may include a word count, search results, a result of sorting the data in a particular manner, and the like. In one embodiment, the DBMS 120 may also update the data stored in the database 105 in response to the query. For example, although not shown in Figure 3, the DBMS 120 may use the database interface 310 to transmit updated database tables 305 to the database 105. In another example, the DBMS 120 may delete or move data stored in the database 105 in response to the query. In another embodiment, the DBMS 120 may store, delete, or move data in the database 105 in response to the client query without returning any results to the client. For example,

the client query may be an instruction to change or update stored data in the database 105 which does not require results to be returned to the client. The DBMS 120 may nonetheless use the database accelerator 145 to, for example, update data that is stored in the database 105 even if the result provided by the database accelerator 145 are stored in the database 105 rather than being delivered to the client.

[0042] Although method 200 illustrates processing a single data block 340 in parallel using the PUs 365, in other embodiments, the database accelerator 145 can process multiple data blocks 340 in parallel. For example, half of the PUs 365 can process one data block 340 while the other half processes a different data block 340. Moreover, in one embodiment, the FPGA 140 can include multiple database accelerators 145 which can operate independently. Thus, one of the accelerators can process one of the data blocks for one of the vertical queues (or threads) shown in Figure 3 while the other accelerator can process a data block from another one of the vertical queues in parallel.

[0043] Figure 4 illustrates converting a database table 305 into multiple data blocks 340 customized for parallel computation, according to an example. For example, the memory in the FPGA implementing the database accelerator can be much smaller than the memory of the CPU hosting the DBMS which may correspond to the size of the database tables 305. For example, the database tables 305 may have a size of 256-512 GB while the memory of the FPGA is limited to 2-8GB. Thus, dividing the database table 305 into the data blocks 340 may permit the FPGA to more efficiently process the data in the tables 305.

[0044] In one embodiment, the database table 305 represents data stored in a relational database. When querying the database, the DBMS may provide a list of rows, columns, or tables that it wants to retrieve from the database. Figure 4 illustrates just one example of a database table 305 where the data is arranged in columns 405 and rows 410 which can be provided to the DBMS by the database. In one embodiment, a data block 340 can include portions of data from multiple database tables 305 by combining and selecting appropriate rows. Generating a data block 340 using selected portions of data from multiple database tables 305 may be useful when referencing multiple tables in other database operations such as a join.

[0045] As mentioned above, the format used to store or arrange the data in the database may not be ideal for analyzing the data using the PUs in the database accelerator which operate in parallel. As such, Figure 4 illustrates performing a conversion 400 where the data in the table 305 is reformatted into the data block 340. In one embodiment, the conversion 400 is performed by the DBMS using a function (e.g., the enqueue computing function) in an accelerator library. However, in another embodiment, the conversion 400 may be performed by logic in the database accelerator- e.g., in hardware in the FPGA.

[0046] In Figure 4, the data block 340 includes respective PU data blocks 420 which, as mentioned above, can be processed by respective ones of the PUs in the database accelerator. In one embodiment, the data block 340 may include a respective PU data block 420 for each PU in the database accelerator although this is not a requirement. For example, some of the PUs may be idle when the data block 340 is processed by the database accelerator, or the PUs which do not have a corresponding PU data block 420 may be used to perform other tasks related to the query.

[0047] Each PU data block 420 includes a header 425 and data The header 425 indicates the number of rows of the data 435 that is in the particular PU data block 420. For example, the PU data blocks 420 may be much smaller than the memory of the CPU hosting the DBMS which may correspond to the size of the database tables 305. For example, the database tables 305 may have a size of 256-512 GB while the memory of the FPGA is limited to 2-8GB. Thus, dividing the database table 305 into the data blocks 340 may permit the FPGA to more efficiently process the data in the tables 305.

[0048] In one embodiment, the database table 305 represents data stored in a relational database. When querying the database, the DBMS may provide a list of rows, columns, or tables that it wants to retrieve from the database. Figure 4 illustrates just one example of a database table 305 where the data is arranged in columns 405 and rows 410 which can be provided to the DBMS by the database. In one embodiment, a data block 340 can include portions of data from multiple database tables 305 by combining and selecting appropriate rows. Generating a data block 340 using selected portions of data from multiple database tables 305 may be useful when referencing multiple tables in other database operations such as a join.

[0049] As mentioned above, the format used to store or arrange the data in the database may not be ideal for analyzing the data using the PUs in the database accelerator which operate in parallel. As such, Figure 4 illustrates performing a conversion 400 where the data in the table 305 is reformatted into the data block 340. In one embodiment, the conversion 400 is performed by the DBMS using a function (e.g., the enqueue computing function) in an accelerator library. However, in another embodiment, the conversion 400 may be performed by logic in the database accelerator - e.g., in hardware in the FPGA.

[0050] In Figure 4, the data block 340 includes respective PU data blocks 420 which, as mentioned above, can be processed by respective ones of the PUs in the database accelerator. In one embodiment, the data block 340 may include a respective PU data block 420 for each PU in the database accelerator although this is not a requirement. For example, some of the PUs may be idle when the data block 340 is processed by the database accelerator, or the PUs which do not have a corresponding

PU data block 420 may be used to perform other tasks related to the query.

[0051] Each PU data block 420 includes a header 425 and data 435. The header 425 indicates the number of rows of the data 435 that is in the particular PU data block 420. For example, the PU data blocks 420 may accelerator 145. Separating the data block 340 into the PU streams 515 takes advantage of the parallelization of the PUs in the database accelerator 145. A PU can process or analyze its respective PU stream 515 independently of the other PUs processing their PU streams 515. Thus, the PU streams 515 can be processed in parallel by the database accelerator 145.

[0052] Once processed, the database accelerator can call the GET_PU_RESULT functions 520 to retrieve the results from processing the PU streams 515. For example, if the query is a word count query, the PUs traverse the PU streams 515 to identify a particular word specified by the client which is to be counted. By calling the GET_PU_RESULT functions 520, the database accelerator can determine the number of occurrences of the word identified by each of the PUs.

[0053] The combiner 375 receives the results from calling the GET_PU_RESULT functions 520 and combines the results. In this manner, the results from the PUs can be merged into combined results 530 of the database accelerator 145 which are stored in an output buffer 525. In one embodiment, the output buffer 525 forwards the combined results the DBMS 120 in response to an instruction from the event manager. For example, the event manager may monitor the database accelerator 145 to determine when all of the PUs have finished processing a particular data block 340 (or blocks). Once complete, the event manager instructs the database accelerator 145 to forward the combined result 530 to the DBMS 120.

[0054] In one embodiment, the functions used to process and transfer the data within the database accelerator 145 can be simulated from software code such as C, C++, Open CL, and the like rather than generating the functions directly from hardware description languages (HDL) or register transfer level (RTL) code. For example, a high-level synthesis (HLS) application can configure a programmable circuit (e.g., an FPGA) using software code without the need of a programmer to manually create RTL code. Moreover, the HLS application can include libraries for handling abstract concepts such different data types (e.g., integers, floating points, etc.), streaming data structures, advanced mathematical formulas (e.g., sine, cosine, square root, etc.), or digital signal processing (e.g., convolution, amplitude modulation, decoders, etc.).

[0055] In one embodiment, the HLS application is used to generate the streaming environment shown in Figure 5. For example, a portion of the data block 340 can be fed into the PU streams 515 with different read and write data widths which can be designed to keep each PU operating in parallel with maximum efficiency. Moreover, a DDR burst length when transmitting the data blocks 340 from the DDR memory 505 to the read buffer 510 can be long enough to reduce DDR inefficiency. A wide DDR width maximizes efficiency with optimal PU resource use and number of processing cycles. In one embodiment, the usage of block RAMs as the read buffer 510 and the output buffer 525 can be reduced by striped read buffer.

[0056] In one embodiment, for a SQL query, the results from the parallel executing PUs can be combined for a single Kernel or Compute Unit on the FPGA. For example, one query 6 processing unit that has a data block size of 2MB and a PU block size of 32KB can be performed with a greater than 25 times improvement of the execution time when compared to performing the query 6 operation on a host CPU. As the number of PUs increase in the database accelerator 145, the execution time can be further improved.

[0057] The descriptions of the various embodiments of the present invention have been presented for purposes of illustration, but are not intended to be exhaustive or limited to the embodiments disclosed. The terminology used herein was chosen to best explain the principles of the embodiments, the practical application or technical improvement over technologies found in the marketplace, or to enable others of ordinary skill in the art to understand the embodiments disclosed herein.

[0058] In the preceding, reference is made to embodiments presented in this disclosure. However, the scope of the present disclosure is not limited to specific described embodiments. Instead, any combination of the features and elements described herein, whether related to different embodiments or not, is contemplated to implement and practice contemplated embodiments. Furthermore, although embodiments disclosed herein may achieve advantages over other possible solutions or over the prior art, whether or not a particular advantage is achieved by a given embodiment is not limiting of the scope of the present disclosure. Thus, the aspects, features, embodiments and advantages described herein are merely illustrative and are not considered elements or limitations of the appended claims except where explicitly recited in a claim(s). Likewise, reference to "the invention" shall not be construed as a generalization of any inventive subject matter disclosed herein and shall not be considered to be an element or limitation of the appended claims except where explicitly recited in a claim(s).

[0059] Aspects described herein may take the form of an entirely hardware embodiment, an entirely software embodiment (including firmware, resident software, micro-code, etc.) or an embodiment combining software and hardware aspects that may all generally be referred to herein as a "module" or "system."

[0060] The present invention may be a system, a method, and/or a computer program product. The computer program product may include a computer readable storage medium (or media) having computer readable program instructions thereon for causing a processor to carry out aspects of the present invention.

[0061] The computer readable storage medium can be a tangible device that can retain and store instructions for use by an instruction execution device. The computer readable storage medium may be, for example, but is not limited to, an electronic storage device, a magnetic storage device, an optical storage device, an electromagnetic storage device, a semiconductor storage device, or any suitable combination of the foregoing. A non-exhaustive list of more specific examples of the computer readable storage medium includes the following: a portable computer diskette, a hard disk, a random access memory (RAM), a read-only memory (ROM), an erasable programmable read-only memory (EPROM or Flash memory), a static random access memory (SRAM), a portable compact disc read-only memory (CD-ROM), a digital versatile disk (DVD), a memory stick, a floppy disk, a mechanically encoded device such as punch-cards or raised structures in a groove having instructions recorded thereon, and any suitable combination of the foregoing. A computer readable storage medium, as used herein, is not to be construed as being transitory signals per se, such as radio waves or other freely propagating electromagnetic waves, electromagnetic waves propagating through a waveguide or other transmission media (e.g., light pulses passing through a fiber-optic cable), or electrical signals transmitted through a wire.

[0062] Computer readable program instructions described herein can be downloaded to respective computing/processing devices from a computer readable storage medium or to an external computer or external storage device via a network, for example, the Internet, a local area network, a wide area network and/or a wireless network. The network may comprise copper transmission cables, optical transmission fibers, wireless transmission, routers, firewalls, switches, gateway computers and/or edge servers. A network adapter card or network interface in each computing/processing device receives computer readable program instructions from the network and forwards the computer readable program instructions for storage in a computer readable storage medium within the respective computing/processing device.

[0063] Computer readable program instructions for carrying out operations of the present invention may be assembler instructions, instruction-set-architecture (ISA) instructions, machine instructions, machine dependent instructions, microcode, firmware instructions, state-setting data, or either source code or object code written in any combination of one or more programming languages, including an object oriented programming language such as Smalltalk, C++ or the like, and conventional procedural programming languages, such as the "C" programming language or similar programming languages. The computer readable program instructions may execute entirely on the user's computer, partly on the user's computer, as a stand-alone software package, partly on the user's computer and partly on a remote computer or entirely on the remote computer or server. In the latter scenario, the remote computer may be connected to the user's computer through any type of network, including a local area network (LAN) or a wide area network (WAN), or the connection may be made to an external computer (for example, through the Internet using an Internet Service Provider). In some embodiments, electronic circuitry including, for example, programmable logic circuitry, field-programmable gate arrays (FPGA), or programmable logic arrays (PLA) may execute the computer readable program instructions by utilizing state information of the computer readable program instructions to personalize the electronic circuitry, in order to perform aspects of the present invention.

[0064] Aspects of the present invention are described herein with reference to flowchart illustrations and/or block diagrams of methods, apparatus (systems), and computer program products according to embodiments of the invention. It will be understood that each block of the flowchart illustrations and/or block diagrams, and combinations of blocks in the flowchart illustrations and/or block diagrams, can be implemented by computer readable program instructions.

[0065] These computer readable program instructions may be provided to a processor of a general purpose computer, special purpose computer, or other programmable data processing apparatus to produce a machine, such that the instructions, which execute via the processor of the computer or other programmable data processing apparatus, create means for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks. These computer readable program instructions may also be stored in a computer readable storage medium that can direct a computer, a programmable data processing apparatus, and/or other devices to function in a particular manner, such that the computer readable storage medium having instructions stored therein comprises an article of manufacture including instructions which implement aspects of the function/act specified in the flowchart and/or block diagram block or blocks.

[0066] The computer readable program instructions may also be loaded onto a computer, other programmable data processing apparatus, or other device to cause a series of operational steps to be performed on the computer, other programmable apparatus or other device to produce a computer implemented process, such that the instructions which execute on the computer, other programmable apparatus, or other device implement the functions/acts specified in the flowchart and/or block diagram block or blocks.

[0067] The flowchart and block diagrams in the Figures illustrate the architecture, functionality, and operation of possible implementations of systems, methods, and computer program products according to various embodiments of the present invention. In this regard, each block in the flowchart or block diagrams may represent a module, segment, or portion of instructions, which comprises one or more executable instructions for implementing the specified logical function(s). In some alternative imple-

mentations, the functions noted in the block may occur out of the order noted in the figures. For example, two blocks shown in succession may, in fact, be executed substantially concurrently, or the blocks may sometimes be executed in the reverse order, depending upon the functionality involved. It will also be noted that each block of the block diagrams and/or flowchart illustration, and combinations of blocks in the block diagrams and/or flowchart illustration, can be implemented by special purpose hardware-based systems that perform the specified functions or acts or carry out combinations of special purpose hardware and computer instructions.

[0068] While the foregoing is directed to specific examples, other and further examples may be devised without departing from the basic scope thereof, and the scope thereof is determined by the claims that follow.

Claims

1. A query processing system (100), comprising:

a host central processing unit, CPU, (115) configured to execute a database management system, DBMS, (120); and
a database accelerator (145) separate from the host CPU (115), wherein the database accelerator (145) comprises a plurality of processing units, PUs, (150),
wherein the DBMS (120) comprises an accelerator library (125) and is configured to:

receive a query to execute on a database (105);
retrieve a database table from the database (105) corresponding to the query;
reformat the database table comprising rows and columns into a data block, wherein the data block comprises a plurality of PU data blocks, each corresponding to one of the plurality of PUs (150), wherein each of the plurality of PU data blocks comprises a portion of data stored in the rows and columns of the database table, wherein reformatting the database table comprises using an enqueue compute function from the accelerator library (125) for converting the data received from the databases (105) into a format that can be processed in parallel by the database accelerator (145);
transmit the data block to the database accelerator (145),

wherein the database accelerator (145) is configured to:

process the plurality of PU data blocks using the plurality of PUs (150) in parallel; and
forward results from the plurality of PUs (150) to the DBMS (120);
wherein each of the PUs has a respective memory element for storing the PU data blocks;

wherein the DBMS (120) is configured to add headers to each of the plurality of PU data blocks indicating a number of rows of data in each of the plurality of PU data blocks; wherein the DBMS is configured to update row indicators in the headers to indicate the amount of data in the respective PU data block when the plurality of PU data blocks have different amount of data.

2. The query processing system of claim 1, wherein the database accelerator (145) comprises a combiner configured to receive individual results from each of the plurality of PUs (150) and combine the individual results into a combined result which is forwarded to the DBMS (120).

3. The query processing system of claim 1, wherein the query comprises a structured query language (SQL) query.

4. The query processing system of claim 1, wherein the database accelerator (145) is hosted on one or more programmable integrated circuits.

5. The query processing system of claim 4, wherein the DBMS (120) comprises a programmable integrated circuit framework that serves as an interface between the DBMS (120) and the one or more programmable integrated circuits, wherein the programmable integrated circuit framework is configured to translate instructions issued by the DBMS (120) into commands that can be interpreted by the one or more programmable integrated circuits.

6. The query processing system of claim 4, wherein the one or more programmable integrated circuits form a field-programmable gate array (FPGA).

7. The query processing system of claim 1, wherein the database accelerator (145) is hosted on one or more application specific integrated circuits (ASICs).

8. A method of processing a query using a database accelerator (145), comprising:

receiving a data block from a DBMS (120) executing on a separate host CPU (115) with the database accelerator (145) having a plurality of processing units, PUs, (150) implemented on one or more integrated circuits, wherein the data block is based on a database table comprising rows and columns retrieved from a database

(105) and comprises a plurality of PU data blocks, each corresponding to one of the plurality of PUs (150), wherein each of the plurality of PU data blocks comprises a portion of data stored in the rows and columns of the database table,
receiving, by the DBMS (120), a query to execute on the database (105);
retrieving, by the DBMS (120), the database table from the database (105) corresponding to the query;
reformatting, by the DBMS (120), the database table into the data block using an enqueue compute function from an accelerator library (125) of the DBMS (120) for converting the data received from the database (105) into a format that can be processed in parallel by the database accelerator (145);
transmitting, by the DBMS (120), the data block to the database accelerator (145);
processing, by the database accelerator (145), the plurality of PU data blocks in parallel; and
receiving individual results from each of the plurality of PUs (150) with a combiner and combining the individual results into a combined result which is forwarded to the DBMS (120);
wherein each of the PUs has a respective memory element for storing the PU data blocks;
wherein each of the plurality of PU data blocks comprises a header indicating a number of rows of data in each of the plurality of PU data blocks wherein the DBMS is configured to update row indicators in the headers to indicate the amount of data in the respective PU data block when the plurality of PU data blocks have different amount of data.

9. The method of claim 8, wherein the database table is retrieved from the database (105) using a SQL query.

10. The method of claim 8, wherein the one or more integrated circuits comprise one or more programmable integrated circuits.

11. The method of claim 10, further comprising: the database accelerator (145) is configured to communicate with a programmable integrated circuit framework in the DBMS (120) that serves as an interface between the DBMS (120) and the one or more programmable integrated circuits; and
wherein the programmable integrated circuit framework is configured to translate instructions issued by the DBMS (120) into commands that can be interpreted by the one or more programmable integrated circuits.

12. The method of claim 10, wherein the one or more programmable integrated circuits form a field-programmable gate array (FPGA).

Patentansprüche

1. Abfrageverarbeitungssystem (100), umfassend:

eine Host-Zentralverarbeitungseinheit, CPU, (115), die konfiguriert ist, um ein Datenbankmanagementsystem, DBMS, (120) auszuführen; und
einen Datenbankbeschleuniger (145), der von der Host-CPU (115) getrennt ist, wobei der Datenbankbeschleuniger (145) eine Vielzahl von Verarbeitungseinheiten, PUs, (150) umfasst,
wobei das DBMS (120) eine Beschleunigerbibliothek (125) umfasst und konfiguriert ist zum:

Empfangen einer Abfrage, um sie auf einer Datenbank (105) auszuführen;
Abrufen einer Datenbanktabelle aus der Datenbank (105), die der Abfrage entspricht;
Neuformatieren der Datenbanktabelle, umfassend Zeilen und Spalten, in einen Datenblock, wobei der Datenblock eine Vielzahl von PU-Datenblöcken umfasst, die jeder einer der Vielzahl von PUs (150) entsprechen, wobei jeder der Vielzahl von PU-Datenblöcken einen Abschnitt von Daten umfasst, die in den Zeilen und Spalten der Datenbanktabelle gespeichert sind, wobei das Neuformatieren der Datenbanktabelle ein Verwenden einer Warteschlangenrechnungsfunktion aus der Beschleunigerbibliothek (125), zum Umwandeln der Daten, die von den Datenbanken (105) empfangen werden, in ein Format umfasst, das durch den Datenbankbeschleuniger (145) parallel verarbeitet werden kann;
Übertragen des Datenblocks an den Datenbankbeschleuniger (145), wobei der Datenbankbeschleuniger (145) konfiguriert ist zum:

Paralleles Verarbeiten der Vielzahl von PU-Datenblöcken unter Verwendung der Vielzahl von PUs (150); und
Weiterleiten von Ergebnissen aus der Vielzahl von PUs (150) an das DBMS (120);
wobei jede der PUs ein jeweiliges Speicherelement zum Speichern der PU-Datenblöcke aufweist;
wobei das DBMS (120) konfiguriert ist, um Header zu jedem der Vielzahl von PU-Datenblöcken hinzuzufügen, die

eine Anzahl von Zeilen von Daten in jedem der Vielzahl von PU-Datenblöcken angeben; wobei das DBMS konfiguriert ist, um Zeilenindikatoren in den Headern zu aktualisieren, um die Datenmenge in dem jeweiligen PU-Datenblock anzugeben, wenn die Vielzahl von PU-Datenblöcken unterschiedliche Datenmenge aufweisen.

2. Abfrageverarbeitungssystem nach Anspruch 1, wobei der Datenbankbeschleuniger (145) einen Kombinierer umfasst, der konfiguriert ist, um einzelne Ergebnisse aus jeder der Vielzahl von PUs (150) zu empfangen und die einzelnen Ergebnisse zu einem kombinierten Ergebnis zu kombinieren, das an das DBMS (120) weitergeleitet wird.

3. Abfrageverarbeitungssystem nach Anspruch 1, wobei die Abfrage eine Abfrage nach Structured Query Language (SQL) umfasst.

4. Abfrageverarbeitungssystem nach Anspruch 1, wobei der Datenbankbeschleuniger (145) auf einem oder mehreren programmierbaren integrierten Schaltkreisen gehostet ist.

5. Abfrageverarbeitungssystem nach Anspruch 4, wobei das DBMS (120) ein Framework für den programmierbaren integrierten Schaltkreis umfasst, das als eine Schnittstelle zwischen dem DBMS (120) und dem einen oder den mehreren programmierbaren integrierten Schaltkreisen dient, wobei das Framework für den programmierbaren integrierten Schaltkreis konfiguriert ist, um Anweisungen, die durch das DBMS (120) ausgegeben werden, in Befehle zu übersetzen, die durch den einen oder die mehreren programmierbaren integrierten Schaltkreise interpretiert werden können.

6. Abfrageverarbeitungssystem nach Anspruch 4, wobei der eine oder die mehreren programmierbaren integrierten Schaltkreise ein Field-Programmable Gate Array (FPGA) ausbilden.

7. Abfrageverarbeitungssystem nach Anspruch 1, wobei der Datenbankbeschleuniger (145) auf einem oder mehreren anwendungsspezifischen integrierten Schaltkreisen (ASICs) gehostet ist.

8. Verfahren zum Verarbeiten einer Abfrage unter Verwendung eines Datenbankbeschleunigers (145), umfassend:

Empfangen eines Datenblocks von einem DBMS (120), das auf einer getrennten Host-CPU (115) mit dem Datenbankbeschleuniger (145) ausgeführt wird, der eine Vielzahl von Verarbeitungseinheiten, PUs, (150) aufweist, die auf einem oder mehreren integrierten Schaltkreisen implementiert sind, wobei der Datenblock auf einer Datenbanktabelle basiert, umfassend Zeilen und Spalten, die aus einer Datenbank (105) abgerufen werden, und eine Vielzahl von PU-Datenblöcken umfasst, die jeder einer der Vielzahl von PUs (150) entsprechen, wobei jeder der Vielzahl von PU-Datenblöcken einen Abschnitt von Daten umfasst, die in den Zeilen und Spalten der Datenbanktabelle gespeichert sind,

Empfangen, durch das DBMS (120), einer Abfrage, um sie auf einer Datenbank (105) auszuführen;

Abrufen, durch das DBMS (120), der Datenbanktabelle aus der Datenbank (105), die der Abfrage entspricht;

Neuformatieren, durch das DBMS (120), der Datenbanktabelle in den Datenblock unter Verwendung einer Warteschlangenrechnungsfunktion aus einer Beschleunigerbibliothek (125) des DBMS (120) zum Umwandeln der Daten, die von der Datenbank (105) empfangen werden, in ein Format, das durch den Datenbankbeschleuniger (145) parallel verarbeitet werden kann;

Übertragen, durch das DBMS (120), des Datenblocks an den Datenbankbeschleuniger (145);

Paralleles Verarbeiten, durch den Datenbankbeschleuniger (145), der Vielzahl von PU-Datenblöcken; und

Empfangen von einzelnen Ergebnissen aus jeder der Vielzahl von PUs (150) mit einem Kombinierer und Kombinieren der einzelnen Ergebnisse zu einem kombinierten Ergebnis, das an das DBMS (120) weitergeleitet wird;

wobei jede der PUs ein jeweiliges Speicherelement zum Speichern der PU-Datenblöcke aufweist;

wobei jeder der Vielzahl von PU-Datenblöcken einen Header umfasst, der eine Anzahl von Zeilen von Daten in jedem der Vielzahl von PU-Datenblöcken angibt,

wobei das DBMS konfiguriert ist, um Zeilenindikatoren in den Headern zu aktualisieren, um die Datenmenge in dem jeweiligen PU-Datenblock anzugeben, wenn die Vielzahl von PU-Datenblöcken unterschiedliche Datenmenge aufweisen.

9. Verfahren nach Anspruch 8, wobei die Datenbanktabelle unter Verwendung einer SQL-Abfrage aus der Datenbank (105) abgerufen wird.

10. Verfahren nach Anspruch 8, wobei der eine oder die mehreren integrierten Schaltkreise einen oder mehrere programmierbare integrierte Schaltkreise um-

fassen.

11. Verfahren nach Anspruch 10, ferner umfassend: den Datenbankbeschleuniger (145), der konfiguriert ist, um mit einem Framework für den programmierbaren integrierten Schaltkreis in dem DBMS (120) zu kommunizieren, das als eine Schnittstelle zwischen dem DBMS (120) und dem einen oder den mehreren programmierbaren integrierten Schaltkreisen dient; und wobei das Framework für den programmierbaren integrierten Schaltkreis konfiguriert ist, um Anweisungen, die durch das DBMS (120) ausgegeben werden, in Befehle zu übersetzen, die durch den einen oder die mehreren programmierbaren integrierten Schaltkreise interpretiert werden können.

12. Verfahren nach Anspruch 10, wobei der eine oder die mehreren programmierbaren integrierten Schaltkreise ein Field-Programmable Gate Array (FPGA) ausbilden.

Revendications

1. Système de traitement de requêtes (100), comprenant :

une unité centrale de traitement, CPU, hôte, (115) configurée pour exécuter un système de gestion de base de données, DBMS, (120) ; et un accélérateur de base de données (145) distinct de la CPU hôte (115), dans lequel l'accélérateur de base de données (145) comprend une pluralité d'unités de traitement, PU, (150), dans lequel le DBMS (120) comprend une bibliothèque d'accélérateurs (125) et est configuré pour :

recevoir une requête à exécuter sur une base de données (105) ;
récupérer un tableau de base de données à partir de la base de données (105) correspondant à la requête ;
reformater le tableau de base de données comprenant des rangées et des colonnes dans un bloc de données, dans lequel le bloc de données comprend une pluralité de blocs de données de PU, chacun correspondant à l'une de la pluralité de PU (150), dans lequel chacun de la pluralité de blocs de données de PU comprend une partie de données stockées dans les rangées et colonnes du tableau de base de données, dans lequel le reformatage du tableau de bases de données comprend l'utilisation d'une fonction de calcul de mise en file d'attente de la bibliothèque d'accélérateurs (125) pour convertir les données reçues en provenance des bases de données (105) en un format qui peut être traité en parallèle par l'accélérateur de base de données (145) ;
transmettre le bloc de données à l'accélérateur de base de données (145), dans lequel l'accélérateur de base de données (145) est configuré pour :

traiter la pluralité de blocs de données de PU à l'aide de la pluralité de PU (150) en parallèle ; et
transférer des résultats de la pluralité de PU (150) au DBMS (120) ;
dans lequel chacune des PU a un élément de mémoire respectif pour stocker les blocs de données de PU ;
dans lequel le DBMS (120) est configuré pour ajouter des en-têtes à chacun de la pluralité de blocs de données de PU indiquant un nombre de rangées de données dans chacun de la pluralité de blocs de données de PU ; dans lequel le DBMS est configuré pour mettre à jour des indicateurs de rangée dans les en-têtes pour indiquer la quantité de données dans le bloc de données de PU respectif lorsque la pluralité de blocs de données de PU ont une quantité de données différente.

2. Système de traitement de requêtes selon la revendication 1, dans lequel l'accélérateur de base de données (145) comprend un combinateur configuré pour recevoir des résultats individuels en provenance de chacune de la pluralité de PU (150) et combiner les résultats individuels en un résultat combiné qui est transféré au DBMS (120).

3. Système de traitement de requêtes selon la revendication 1, dans lequel la requête comprend une requête de langage de requêtes structurées (SQL).

4. Système de traitement de requêtes selon la revendication 1, dans lequel l'accélérateur de base de données (145) est hébergé sur un ou plusieurs circuits intégrés programmables.

5. Système de traitement de requêtes selon la revendication 4, dans lequel le DBMS (120) comprend un cadre de circuit intégré programmable qui sert d'interface entre le DBMS (120) et le ou les circuits intégrés programmables, dans lequel le cadre de circuit intégré programmable est configuré pour traduire des instructions émises par le DBMS (120) en commandes qui peuvent être interprétées par le ou les circuits intégrés programmables.

6. Système de traitement de requêtes selon la revendication 4, dans lequel le ou les circuits intégrés programmables forment une matrice prédiffusée programmable par l'utilisateur (FPGA).

7. Système de traitement de requêtes selon la revendication 1, dans lequel l'accélérateur de base de données (145) est hébergé sur un ou plusieurs circuits intégrés spécifiques à une application (ASIC).

8. Procédé de traitement d'une requête à l'aide d'un accélérateur de base de données (145), comprenant :

la réception d'un bloc de données en provenance d'un DBMS (120) s'exécutant sur une CPU hôte (115) distincte avec l'accélérateur de base de données (145) ayant une pluralité d'unités de traitement, PU, (150) mises en oeuvre sur un ou plusieurs circuits intégrés, dans lequel le bloc de données est basé sur un tableau de base de données comprenant des rangées et des colonnes récupérées à partir d'une base de données (105) et comprend une pluralité de blocs de données de PU, correspondant chacun à l'une de la pluralité de PU (150), dans lequel chacun de la pluralité de blocs de données de PU comprend une partie de données stockées dans les rangées et colonnes du tableau de base de données,
la réception, par le DBMS (120), d'une requête à exécuter sur la base de données (105) ;
la récupération, par le DBMS (120), du tableau de base de données à partir de la base de données (105) correspondant à la requête ;
le reformatage, par le DBMS (120), du tableau de base de données en bloc de données à l'aide d'une fonction de calcul de mise en file d'attente provenant d'une bibliothèque d'accélérateurs (125) du DBMS (120) pour convertir les données reçues en provenance de la base de données (105) en un format qui peut être traité en parallèle par l'accélérateur de base de données (145) ;
la transmission, par le DBMS (120), du bloc de données à l'accélérateur de base de données (145) ;
le traitement, par l'accélérateur de base de données (145), de la pluralité de blocs de données de PU en parallèle ; et
la réception de résultats individuels en provenance de chacune de la pluralité de PU (150) avec un combinateur et la combinaison des résultats individuels en un résultat combiné qui est transféré au DBMS (120) ;
dans lequel chacune des PU a un élément de mémoire respectif pour stocker les blocs de données de PU ;
dans lequel chacun de la pluralité de blocs de données de PU comprend un en-tête indiquant un nombre de rangées de données dans chacun de la pluralité de blocs de données de PU
dans lequel le DBMS est configuré pour mettre à jour des indicateurs de rangée dans les entêtes pour indiquer la quantité de données dans le bloc de données de PU respectif lorsque la pluralité de blocs de données de PU ont une quantité de données différente.

9. Procédé selon la revendication 8, dans lequel le tableau de base de données est récupéré à partir de la base de données (105) à l'aide d'une requête SQL.

10. Procédé selon la revendication 8, dans lequel le ou les circuits intégrés comprennent un ou plusieurs circuits intégrés programmables.

11. Procédé selon la revendication 10, comprenant en outre : l'accélérateur de base de données (145) configuré pour communiquer avec un cadre de circuit intégré programmable dans le DBMS (120) qui sert d'interface entre le DBMS (120) et le ou les circuits intégrés programmables ; et
dans lequel le cadre de circuit intégré programmable est configuré pour traduire des instructions émises par le DBMS (120) en commandes qui peuvent être interprétées par le ou les circuits intégrés programmables.

12. Procédé selon la revendication 10, dans lequel le ou les circuits intégrés programmables forment une matrice prédiffusée programmable par l'utilisateur (FPGA).

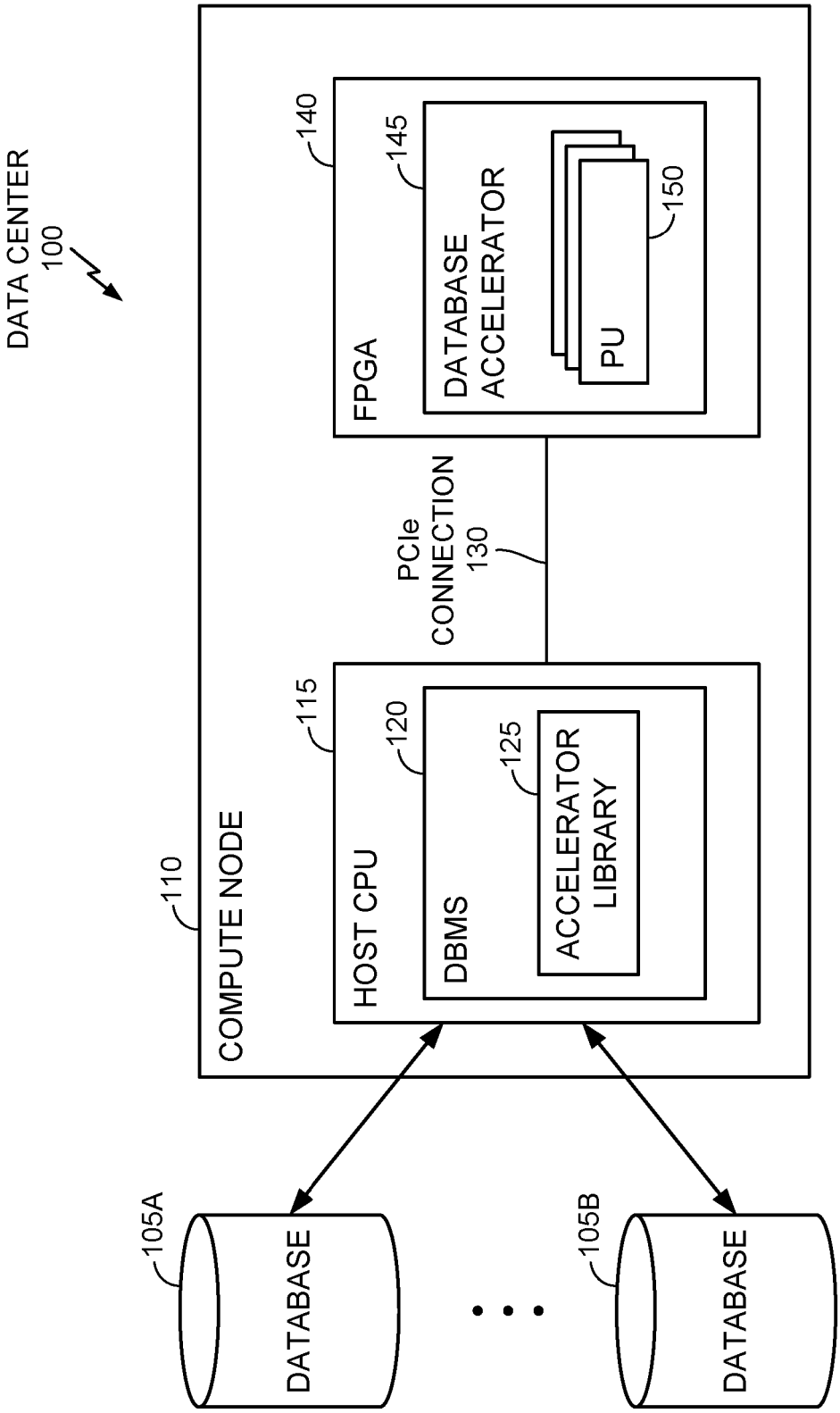
DATA CENTER
100
COMPUTE NODE
110
HOST CPU
115
DBMS
120
ACCELERATOR LIBRARY
125
PCIe CONNECTION
130
FPGA
140
DATABASE ACCELERATOR
145
PU
150
105A
DATABASE
105B
DATABASE

FIG. 1

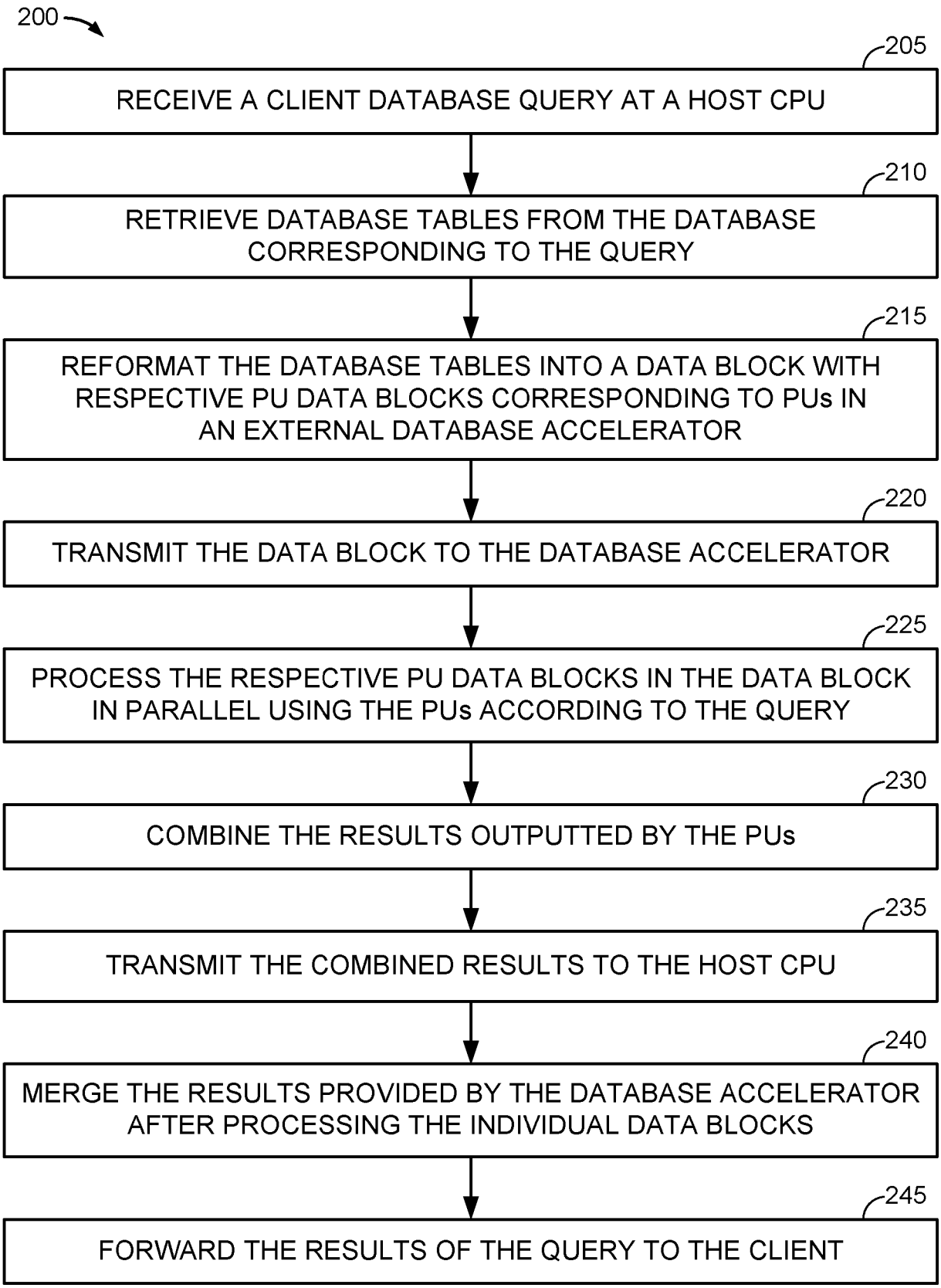
200
205
RECEIVE A CLIENT DATABASE QUERY AT A HOST CPU
210
RETRIEVE DATABASE TABLES FROM THE DATABASE CORRESPONDING TO THE QUERY
215
REFORMAT THE DATABASE TABLES INTO A DATA BLOCK WITH RESPECTIVE PU DATA BLOCKS CORRESPONDING TO PUs IN AN EXTERNAL DATABASE ACCELERATOR
220
TRANSMIT THE DATA BLOCK TO THE DATABASE ACCELERATOR
225
PROCESS THE RESPECTIVE PU DATA BLOCKS IN THE DATA BLOCK IN PARALLEL USING THE PUs ACCORDING TO THE QUERY
230
COMBINE THE RESULTS OUTPUTTED BY THE PUs
235
TRANSMIT THE COMBINED RESULTS TO THE HOST CPU
240
MERGE THE RESULTS PROVIDED BY THE DATABASE ACCELERATOR AFTER PROCESSING THE INDIVIDUAL DATA BLOCKS
245
FORWARD THE RESULTS OF THE QUERY TO THE CLIENT

FIG. 2

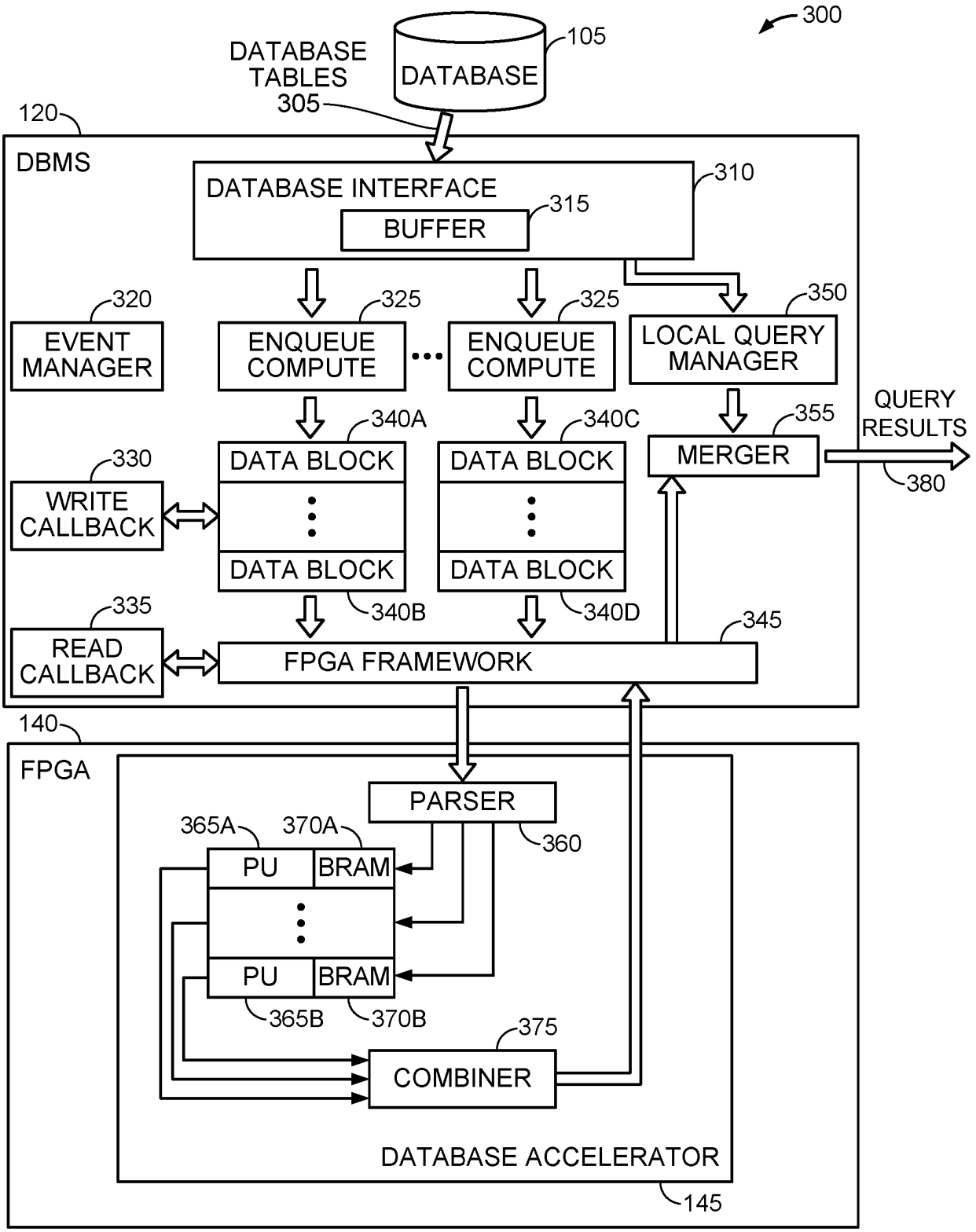
300
DATABASE TABLES
305
DATABASE
105
120
DBMS
DATABASE INTERFACE
310
BUFFER
315
320
EVENT MANAGER
325
ENQUEUE COMPUTE
325
ENQUEUE COMPUTE
350
LOCAL QUERY MANAGER
QUERY RESULTS
340A
340C
355
330
WRITE CALLBACK
DATA BLOCK
DATA BLOCK
DATA BLOCK
DATA BLOCK
MERGER
380
335
340B
340D
345
READ CALLBACK
FPGA FRAMEWORK
140
FPGA
PARSER
360
365A
370A
PU
BRAM
PU
BRAM
365B
370B
375
COMBINER
DATABASE ACCELERATOR
145

FIG. 3

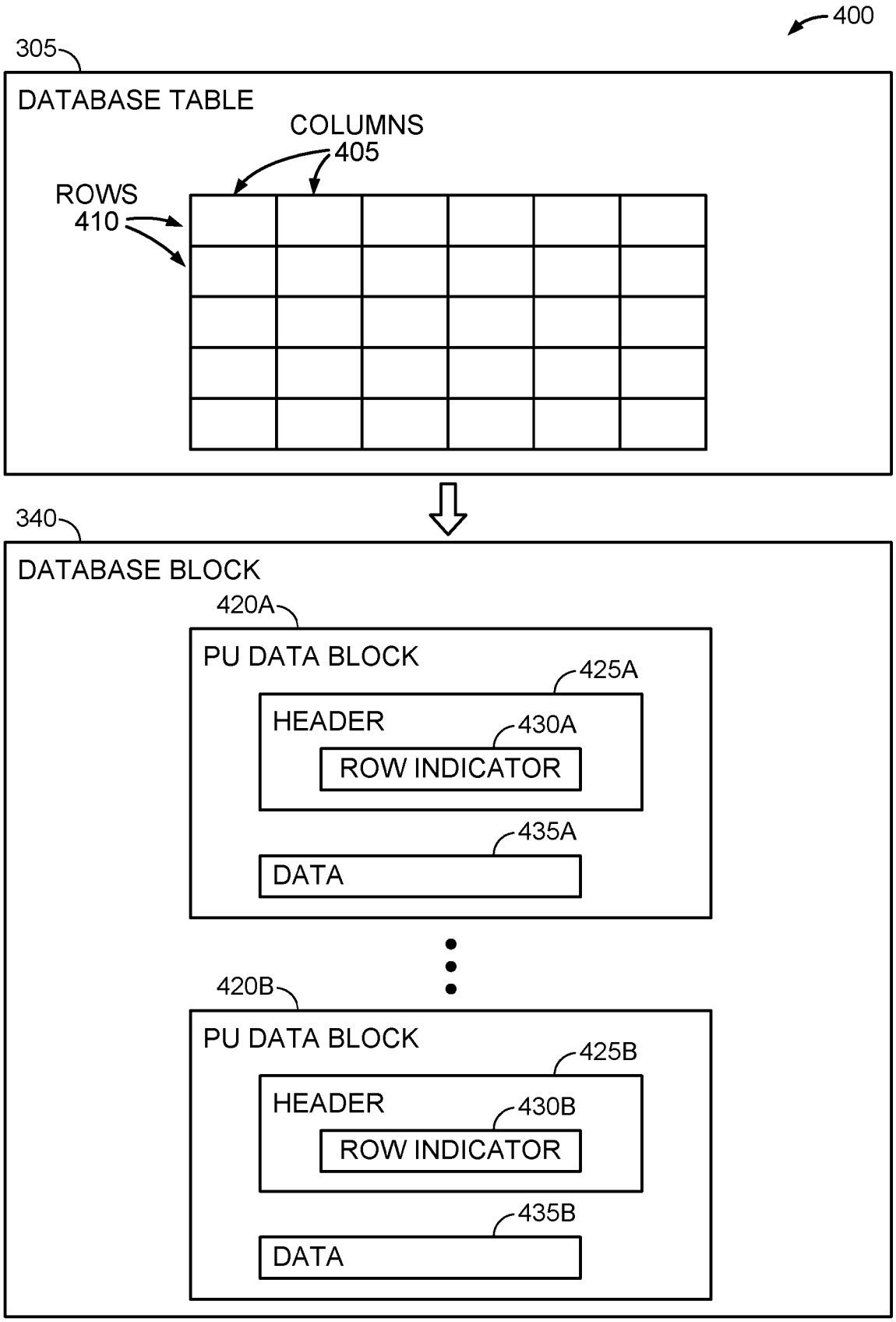
400
305
DATABASE TABLE
COLUMNS
405
ROWS
410
340
DATABASE BLOCK
420A
PU DATA BLOCK
425A
HEADER
430A
ROW INDICATOR
435A
DATA
420B
PU DATA BLOCK
425B
HEADER
430B
ROW INDICATOR
435B
DATA

FIG. 4

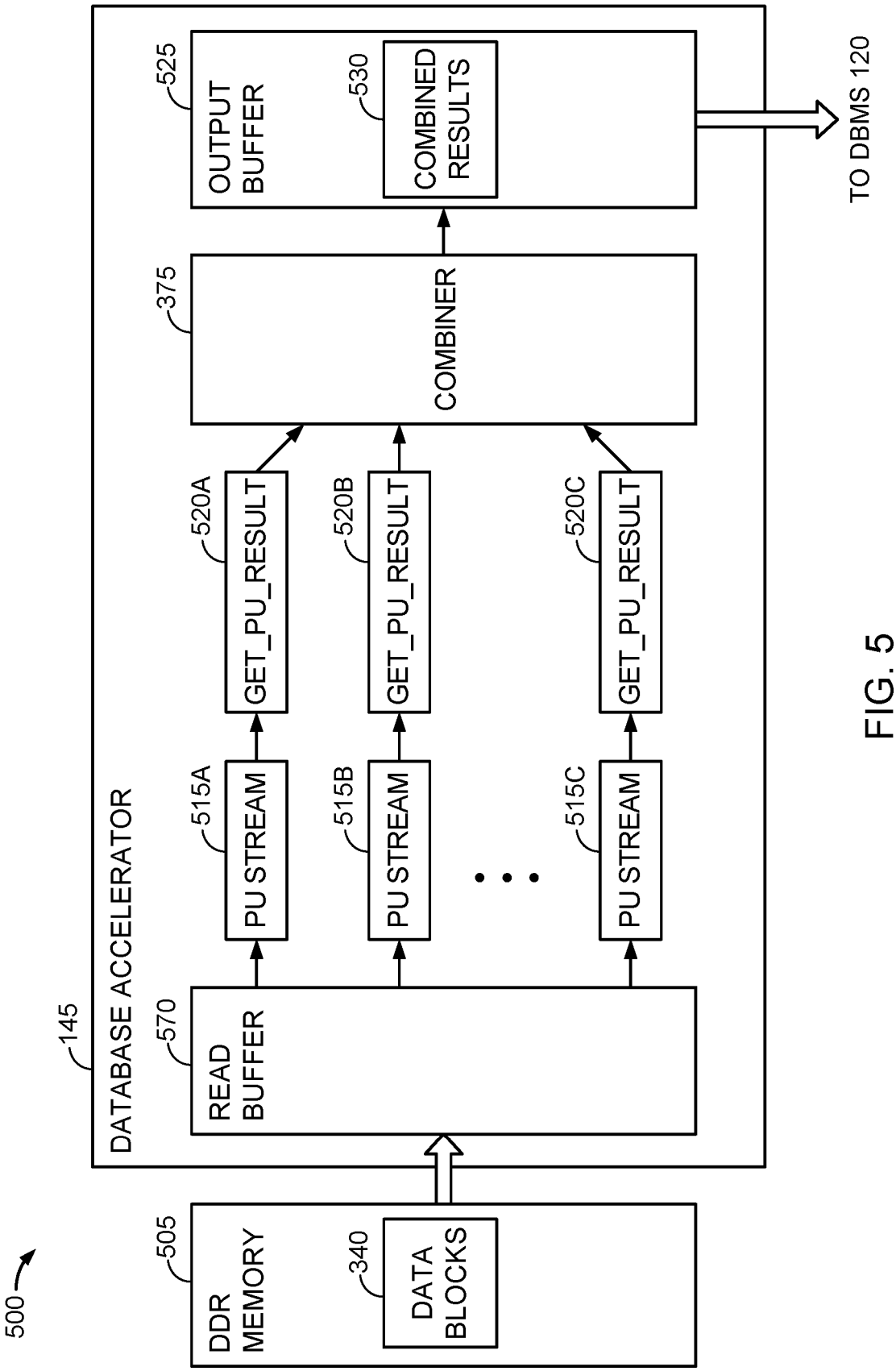
500
DDR MEMORY
505
DATA BLOCKS
340
DATABASE ACCELERATOR
145
READ BUFFER
570
PU STREAM
515A
PU STREAM
515B
PU STREAM
515C
GET_PU_RESULT
520A
GET_PU_RESULT
520B
GET_PU_RESULT
520C
COMBINER
375
OUTPUT BUFFER
525
COMBINED RESULTS
530
TO DBMS 120

FIG. 5

REFERENCES CITED IN THE DESCRIPTION

This list of references cited by the applicant is for the reader's convenience only. It does not form part of the European patent document. Even though great care has been taken in compiling the references, errors or omissions cannot be excluded and the EPO disclaims all liability in this regard.

Non-patent literature cited in the description

- **SALAMI BEHZAD et al.** AxleDB: A novel programmable query processing platform on FPGA. *MICROPROCESSORS AND M ICROSYSTEMS,* vol. 51, ISSN 0141-9331, 142-164 **[0003]**
- Glacier a query-to-hardware compiler. **MUELLER RENE et al.** USER INTERFACE SOFTWARE AND TECHNOLOGY. ACM, 06 June 2010, 159-1 162 **[0004]**
- FPGA-Based Dynamically Reconfigurable SQL Query Processing. **DANIEL ZIENER et al.** ACM TRANSACTIONS ON RECONFIGURABLE TECHNOLOGY AND SYSTEMS. ACM, 22 August 2016, vol. 9, 1-24 **[0005]**
- Energy-aware SQL query acceleration through FPGA-based dynamic partial reconfiguration. **BECHER ANDREAS et al.** 24TH INTERNATIONAL CONFERENCE ON FIELD PROGRAMMABLE LOGIC AND APPLTCATIONS (FPL). TECHNICAL UNIVERSITY OF MUNICH (TUM), 02 September 2014, 1-8 **[0006]**
- **TAVINOR EDOUARD.** *Pipelining for FPGAs. I thought I'd write an explanation of... by Edouard Tavinor I Medium,* 30 March 2017, URL: https://ehrt74.medium.com/pipeliningfor-fpgas-9342d3a2tdl8 **[0008]**